SYSTEM AND METHOD FOR MEASURING DATA NETWORK QUALITY

[001] CLAIM TO PRIORITY

[002] This application claims the benefit of our co-pending United States provisional patent application entitled "METHOD AND SYSTEM TO MEASURE PACKET SWITCHED NETWORK QUALITY" filed April 16, 2003 and assigned serial number 60/463,455, which is incorporated by reference herein.

[003] BACKGROUND OF THE DISCLOSURE

[004] 1. Field of the Invention

[005] The invention relates to a system and a method for measuring the quality of a data network. More specifically, the invention relates to a system and a method for measuring the quality of a data network using a client computer and a server computer communicating using a connection-less transmission protocol.

[006] 2. Description of the Prior Art

[007] A method to measure data networks is known from Iperf (http://dast.nlanr.net/Projects/Iperf). Iperf uses TCP and UDP to measure the performance of an IP network. With UDP, Iperf is able to measure packet loss and delay

jitter.

[008] Another method to measure data networks is known from rfc792. Rfc792 describes the Internet Control Message Protocol (ICMP). With ICMP, it is possible to send measurement data packets between two computers. A first computer sends an ICMP packet containing an originate timestamp to a second computer. The second computer adds a receive timestamp to the ICMP packet when it receives the packet and adds a transmit timestamp to the ICMP packet when it sends the packet back to the first computer.

[009] Aim of the invention

[0010] The aim of the invention is to provide an alternative system and method for measuring the quality of a data network.

[0011] SUMMARY OF THE INVENTION

[0012] The present invention provides an alternative solution for measuring the quality of a data network.

[0013] According to an aspect of the invention a system is provided for measuring the quality of a data network, the system comprising a client computer and a server computer, the client computer and the server computer being arranged to communicate via the data network using a

connection-less transmission protocol, e.g., UDP/IP.

[0014] The client computer can be arranged to send one or more data packets to the server computer. The client computer can comprise a client processor connected to a client memory, the client processor being arranged to read from the client memory, for each data packet, a client interval value, a client packet size value and a packet type value. The client memory can be a file stored on a disk, making it possible to permanently store the information. The packet type value can be a representation of "normal packet" or "burst packet", the latter enabling ~~later on~~ different network quality calculations for different ~~type~~ types of packets. The data packets can comprise a first timestamp indicating the time of sending. The client computer can be arranged to send the data packets with a client interval time equal to the client interval value, the data packets comprising the packet type value and a client payload, the client payload filling the data packets up to a first predefined size equal to the client packet size value. The client payload can be a predefined bit pattern or a random bit pattern. Each data packet can comprise a sequence number, enabling the identification of the order of the packets.

[0015] The server computer can be arranged to receive at least one of the data packets. The server computer can comprise a server processor connected to a server memory.

The server processor can be arranged to read from the server memory a timeout value indicating the predefined amount of time, a server interval value, a quantity value or a server packet size value. The server computer can further be arranged to send, for each received data packet, a modified data packet to the client computer, the modified data packet comprising the first timestamp and a second timestamp indicating the time of receiving. The modified data packet can comprise a server payload, the server payload filling the modified data packet up to a second predefined size equal to the server packet size value. This enables measuring asymmetric network load where uplink and downlink have different packet sizes. The server payload can be a predefined bit pattern or a random bit pattern. The server computer can be arranged to send one or more timeout data packets to the client computer when a predefined amount of time ~~lapsed~~ elapsed since receiving the last data packet in the server computer. This is advantageous because it increases the detectability of lost packets. The server computer can be arranged to send the timeout data packets with a server interval time equal to the server interval value, until a next data packet is received. The maximum number of timeout data packets ~~send~~ sent can be equal to the quantity value. This enables a controlled detection of lost packets.

[0016] The client computer can be arranged to send a configuration packet to the server. The configuration

packet can comprise the timeout value, the server interval value, the quantity value and/or the server packet size value. This can be advantageous because the same client computer using the same connection-less transmission protocol can be used. For configuration, the client computer can be arranged to communicate with the server computer via the data network using a connection-oriented transmission protocol, e.g., TCP/IP. This can be advantageous when it has to be guaranteed that the server computer correctly receives configuration data ~~correctly~~. The server computer is arranged to receive the configuration packet. The server processor can be arranged to write to the server memory the timeout value, the server interval value, the quantity value and/or the server packet size value, so it becomes immediately available for use~~usage immediately~~.

[0017] The client computer can be arranged to store a ~~logfile~~ log file. The ~~logfile~~ log file can comprise all modified data packets received from the server computer. The ~~logfile~~ log file can further comprise all timeout data packets received from the server computer. All modified data packets received from the server computer can be stored in a first ~~logfile~~ log file. All timeout data packets received from the server computer can be stored in a second ~~logfile~~ log file. Each data packet can be stored in a ring buffer. The client computer can be arranged to check each modified data packet received from the server

with the contents of the ring buffer. The client computer can be arranged to delete the data packet from the ring buffer if the corresponding modified data packet is not received within the length of the ring buffer, and in this case store the data packet in the second ~~logfile~~ log file.

[0018] The client computer can arranged to calculate a ~~round_trip time~~round-trip time by subtracting the first timestamp from a timestamp indicating when the modified data packet is received in the client computer. The client computer can be arranged to store the ~~round-trip time~~round-trip time in the first ~~logfile~~log file, enabling later usage of the data.

[0019] The client computer can be arranged to calculate a ~~client_to_server latency~~client-to-server latency by subtracting the second timestamp from the first timestamp. The client computer can be arranged to store the ~~client-to-server latency~~client-to-server latency in the first ~~logfile~~log file, enabling its later usage ~~if the data~~.

[0020] The client computer can be arranged to read from the first ~~logfile~~log file the packet type value, the ~~client_to_server latency~~client-to-server latency and the ~~round_trip time~~round-trip time. The client computer can be arranged to calculate, for each modified data packet with the packet type value equal to the representation of "normal packet", a ~~server-to-client latency~~server-to-client

latency by subtracting the ~~client-to-server latency~~client-to-server latency from the ~~round-trip time~~round-trip time.

[0021] The client computer can be arranged to read from the first ~~logfile~~log file the packet type value. The client computer can be arranged to calculate a downlink bandwidth using:

(#BytesPerPacket+Overhead)*(#BurstPackets)/(t2-t1)

where #BytesPerPacket equals the length of the modified data packet in bytes; Overhead equals the length of protocol header overhead in bytes; #BurstPackets equals the number of modified data packets stored in the first ~~logfile~~log file with the packet type value equal to the representation of "burst packet"; t1 equals a timestamp indicating when the first modified data packet is received in the client computer; t2 equals a timestamp indicating when the last modified data packet is received in the client computer.

[0022] The client computer can be arranged to read from the first ~~logfile~~log file the packet type value. The client computer can be arranged to calculate an uplink bandwidth using:

(#BytesPerClientPacket+Overhead)*(#BurstPackets)/

((t2+t4)-(t1+t3))

where #BytesPerClientPacket equals the length of the data packet in bytes; Overhead equals the length of protocol header overhead in bytes; #BurstPackets equals the number of modified data packets stored in the first ~~logfile~~log file with the packet type value equal to the representation of "burst packet"; t1 equals a timestamp indicating when the first modified data packet is received in the client computer; t2 equals a timestamp indicating when the last modified data packet is received in the client computer; t3 equals the ~~client-to-server latency~~client-to-server latency of the first modified data packet; t4 equals the ~~client-to-server latency~~client-to-server latency of the last modified data packet.

[0023] The client computer can be arranged to read from the first ~~logfile~~log file the ~~round-trip-time~~round-trip time and the sequence number. The client computer can be arranged to calculate a ~~roundtrip~~ round-trip outage using:

if(Rtt2-Rtt1)>t5 then ~~roundtrip~~round-trip outage=true where Rtt2 equals the ~~round-trip-time~~round-trip time; Rtt1 equals the ~~round-trip-time~~round-trip time of the previous modified data packet, the client computer being arranged to identify the previous modified data packet by the sequence number; t5 equals a predefined amount of time.

[0024] The client computer can be arranged to read from the first ~~logfile~~log file the ~~client-to-server latency~~client-to-server latency and the sequence number. The client computer can be arranged to calculate a downlink outage using:

if(Rtt4-Rtt3)>t5 then downlink outage=true

where Rtt4 equals the ~~client-to-server latency~~client-to-server latency; Rtt3 equals the ~~client-to-server latency~~client-to-server latency of the previous modified data packet, the client computer being arranged to identify the previous modified data packet by the sequence number; t5 equals a predefined amount of time.

[0025] The client computer can be arranged to read the sequence number from the first ~~logfile~~log file ~~the sequence number~~. The client computer can be arranged to calculate an uplink outage using:

if(Rtt6-Rtt5)>t5 then uplink outage=true

where Rtt6 equals the ~~server-to-client latency~~server-to-client latency; Rtt5 equals the ~~server-to-client latency~~server-to-client latency of the previous modified data packet, the client computer being arranged to identify the previous modified data

packet by the sequence number; t5 equals a predefined amount of time.

[0026] The client computer can be arranged to determine a number of lost packets between two modified data packets, by analyzing the sequence numbers. This ~~enable~~ provides a good overview of packet loss.

[0027] The client computer can be arranged to combine the calculated values with low-level measurement data from a mobile telephone and/or geographic information data. This is advantageous when poor data network quality is to be indicated in geographical areas, simplifying a search for weak spots in the data network.

[0028] According to an aspect of the invention a method is provided for measuring the quality of a data network in a system comprising a client computer and a server computer, the client computer and the server computer being arranged to communicate via the data network using a connection-less transmission protocol. The method can comprise one or more of the steps of
sending one or more data packets from the client computer to the server computer, the data packets comprising a first timestamp indicating the time of sending, receiving at least one of the data packets in the server computer, sending, for each received data packet, a modified data packet from the server computer to the client computer, the

modified data packet comprising the first timestamp and a second timestamp indicating the time of receiving, sending one or more timeout data packets from the server computer to the client computer when a predefined amount of time ~~lapsed~~ elapsed since receiving the last data packet in the server computer, reading from a server memory in the server computer a timeout value indicating the predefined amount of time, a server interval value and a quantity value, sending the timeout data packets with a server interval time equal to the server interval value, until a next data packet is received, and the maximum number of timeout data packets equals the quantity value, reading from a client memory in the client computer for each data packet a client interval value, a client packet size value and a packet type value, sending the data packets with a client interval time equal to the client interval value, the data packets comprising the packet type value and a client payload, the client payload filling the data packets up to a first predefined size equal to the client packet size value.

[0029] BRIEF DESCRIPTION OF THE DRAWINGS

[0030] The invention will be explained in greater detail by reference to exemplary embodiments shown in the drawings, in which:

[0031] Figure 1 shows an architectural overview of the system;

[0032] Figure 2 shows a data packet or a modified data packet and its elements;

[0033] Figure 3 shows a ~~control~~ configuration packet and its elements;

[0034] Figure 4 shows a time sequence diagram of packets sent between a client computer and a server computer.

[0035] DETAILED DESCRIPTION OF THE INVENTION

[0036] For the purpose of teaching of the invention, preferred embodiments of the method and system of the invention are described in the sequel. It will be apparent to the person skilled in the art that other alternative and equivalent embodiments of the invention can be conceived and reduced to practice without departing from the true spirit of the invention, the scope of the invention being only limited by the claims as finally granted.

[0037] The measurement system consists of a client-server approach. The invention is explained for usage in a GPRS network, but can also be used in any other packet switched network, such as, among others, UMTS, LAN, WAN and Internet. As a connection-less transmission protocol, the UDP/IP protocol is used in the examples. Other connection-less transmission protocols could just as well

have been used.

[0038] Generally speaking and as depicted in Fig. 1, the inventive system for measuring the quality of data network 3 has client computer 1 and server computer 2, with the client and the server computers being arranged to communicate via data network 3 using a connection-less transmission protocol, e.g., UDP/IP.

[0039] Client computer 1 sends one or more data packets to the server computer. The client computer contains client processor 1a connected to client memory 1b. The client processor reads, from the client memory and for each data packet, a client interval value, a client packet size value and a packet type value. Client memory 1b can be a file stored on a disk in order to permanently store the information. As shown in Fig. 2, each such data packet contains the following fields: packet data type value field 101, sequence number field 102, first timestamp field 103, client payload field 104, second timestamp field 105 and server payload field 106. The packet type value signifies that the corresponding packet is either a "normal packet" or a "burst packet", thus enabling different network quality calculations for different types of packets. The first timestamp (field 103) indicates a time at which the data packet was sent. The client computer sends the data packets with an interval time (i.e., time between two such successive packets) equal to

the client interval value. The client payload (field 104) fills the data packet to a first predefined size equal to the client packet size value. The client payload can be a predefined bit pattern or a random bit pattern. The sequence number (field 102) specifically identifies a particular packet relative to and in a sequence of such packets.

[0040] Server computer 2, shown in Fig. 1, receives at least one of the data packets. The server computer contains server processor 2a connected to server memory 2b. The server processor reads, from the server memory, a time-out value indicating a predefined amount of time, a server interval value, a quantity value and a server packet size value. The server computer sends, for each received data packet from the client computer, a modified data packet back to the client computer and which includes the four fields in the data packet itself (fields 101, 102, 103 and 104) and two additional fields -- all collectively as shown in Fig. 2, with the latter two fields being: the second timestamp (field 105) which indicates a time at which the data packet was received at the server and the server payload (field 106) which fills the modified data packet to a second predefined size equal to the server packet size value. Use of the client and server payload fields enables measurement of asymmetric network loading where uplink and downlink have different packet sizes. The server payload can be a predefined bit pattern or a random

bit pattern. Server computer 2 sends one or more time-out data packets to the client computer after a predefined amount of time has elapsed since the time it received its most recent data packet. This is advantageous because it increases the detectability of lost packets. The server computer sends successive time-out data packets with an interval time (here being the time between two such successive packets) equal to the server interval value, until a next data packet is received. The maximum number of the time-out data packets that is so sent in any one instance is given by the quantity value. This enables a controlled detection of lost packets.

[0041] The client computer sends a configuration packet to the server. The configuration packet, as shown in Fig. 3, contains the following fields: time-out value field 201, server interval value field 202, quantity value field 203 and server packet size value field 204. This can be advantageous because the same client computer, through the same connection-less transmission protocol, can be used. Alternatively, for configuration, the client computer can communicate with the server computer via the data network using a connection-oriented transmission protocol, e.g., TCP/IP. This can be advantageous in those instances where when it has to be guaranteed that the server computer will correctly receive configuration data. After the server computer receives the configuration packet, it writes, to the server memory, the time-out value, the server interval

value, the quantity value and the server packet size value from the configuration packet, so that these values are immediately available for use.

[0042] Fig. 4 depicts, in accordance with the present invention, a typical sequence of packets sent between client computer 1 and server computer 2. First, the client computer sends, as symbolized by reference numeral 1001, data packets to the server computer which, in turn, modifies each of the packets and sends a corresponding modified data packet back, the latter operation being symbolized by reference numeral 1002. These operations then continue over time as indicated by reference numeral 1004. Eventually, after a most recent data packet (1001) has been received by server computer 2 and a time-out interval has expired, the server computer sends time-out data packets, as symbolized by reference numeral 1003, to the client computer. In response, the client computer can send further data packets or configuration packets back to the server computer, with these operations being symbolized by respective reference numerals 1001 and 1005.

~~[0038]~~ [0043] The client computer stores a log file, illustratively having two portions such as separate first and second log files. The log file can contain, in the first portion, all modified data packets received from the server computer and, in the second portion, all time-out data packets received from the server computer. Each data

packet, as transmitted by the client computer, can be stored in a ring buffer. The client computer checks each modified data packet received from the server with the contents of the ring buffer and, when the corresponding modified data packet is not received within a length of the ring buffer, deletes the associated data packet from the ring buffer and stores that data packet in the second portion of the log file.

~~[0039]~~[0044] ~~In this example~~ Illustratively, the client computer is a laptop computer in a car connected via a GPRS phone to the GPRS network. Other clients can be used. The client computer ~~It~~ sends a defined pattern of UDP packets to the server computer. The server responses to each data packet with another packet (a modified data packet).

~~[0040]~~[0045] UDP is a OSI level 4 protocol which leaves the IP behavior as transparent as possible. Unlike TCP (the other level well known 4 protocol) UDP does not restrict transmission of packets or force retransmissions in case of missed packets. By using~~Using~~ UDP a level 7 application can keep full contact with the IP layer.

~~[0041]~~[0046] The UDP server is located on a fixed position in the ~~datanetwork~~data network to be tested. For example~~E.g.~~, the server 2 can be directly connected to an Access Point of a GGSN to avoid measurement inaccuracies. Later, the server was connected to the Internet, which

introduced only a few milliseconds extra delay which is ~~neglectible~~ negligible with respect to the delay in the GPRS system.

~~[0042]~~ [0047] The server responds to each packet with the same packet, extended with its time of arrival and possibly filled up with a predefined extra payload.

~~[0043]~~ [0048] In case the server ~~doesn't~~ does not receive any UDP-packets within a certain time after reception of the last packet, it starts sending a predefined number of special packets (timeout packets). Reception of these server-originated packets gives an indication to the client that the upstream stopped, and the duration of the upstream interruption.

~~[0044]~~ [0049] The following server parameters can be remotely configured, using a secure TCP connection with the server on the same port:

~~[0045]~~ [0050] the amount of octets to which it shall fill up the reflected UDP-packets;

~~[0046]~~ [0051] the time before it starts sending its own packets to indicate the interruption of client packets;

~~[0047]~~ [0052] the number of own packets after interruption of client packets.

~~[0048]~~ [0053] The measurement logging is done by two ~~logfiles~~log files: the A-~~logfile~~log file (second ~~logfile~~log file) and the R-~~logfile~~log file (first ~~logfile~~log file). The R-~~logfile~~log file is filled with all packets that returned from the server. The content of the A-file consists of general information on the measurement, like the used packet pattern, as well as exceptions, which have been recorded by the client, like lossed packets.

~~[0049]~~ [0054] The client stores every sent packet in a ~~ringbuffer~~ring buffer (not specifically shown), and checks every received packet which the contents of the ~~ringbuffer~~ring buffer. If the packet is recognized, it is stored in the R-result file, together with its round trip time and the one-way delay time. The latter is only usable if client and server clocks are synchronized.

~~[0050]~~ [0055] In case the packet is unknown the packet content is stored in the A-log file. This happens for example in case of reception of a server originated packet.

~~[0051]~~ [0056] If a packet did not return within the length of the ~~ringbuffer~~ring buffer, it is deleted from the buffer and stored in the ~~A logfile~~A-log file as 'lost'. Packets which are not marked as 'returned' when the measurement stops are also stored in the ~~A logfile~~A-log file.

~~[0052]~~ [0057] Server part

~~[0053]~~ [0058] The server listens on a defined port for both UDP packets and TCP connect requests. The port number is given as a parameter in the command line to boot the server.

~~[0054]~~ [0059] In case of reception of a UDP packet:

~~[0055]~~ [0060] If the first character of the client packet is a 'C' (i.e., this is a control data packet) and the characters from the 19th until the 30th position are valid numbers the packet is processed as follows:

Syntax of the command packet: CYYYYMMDDHHmmSSNNNiiiigggnnn
Example: C20030422105256873012507000050
Where:
YYYY: year, 2003
MM: month, 04
DD: day, 22
HH: hour, 10
mm: minute, 52
SS: second, 56
NNN: millisecond, 873
iiii: interval between the packets, 125 ms
gggg: packet size, 700 octets
nnn: number of packets, 50

~~[0056]~~ [0061] The date and time stamp of the packet reflect its actual transmission time, derived from the client's clock. After reception of this packet, the server starts the server-originated packet sequence according to the parameters iiii, gggg and nnn. After sending the last packet, the server returns to a standby state. The payload of the packets sent by the server contains the RTT's (round-trip times) of the last measurement session.

~~[0057]~~ [0062] The server remains alert for other commands during the transmission. These commands might override former commands. Example: if the server is in the state of transmitting 1000 packets with an interval of 1000 ms, the client may send a new command requesting 1 packet. Then only this single packet will be sent by the server and the other packets will not be sent.

~~[0058]~~ [0063] If the received UDP packet does not start with the character 'C' the packet is sent back to the client, extended with a timestamp which indicates the time on which the server received the packet. It is coded in the same way: YYYYMMDDHHmmSSNNN.

~~[0059]~~ [0064] In case the packet size is less than the requested size, the packet is filled up to the requested size, using a sequence of the character 'A' (predefined bit pattern).

~~[0060]~~ [0065] After reception of a packet like this, a timer is started. This timer can be remotely configured using the HW command (see the TCP section below). If no new packet is received before the timer expires, the server starts sending server-originated packets with a packet size, to be configured with the HE command. The packets are coded like:

SqqqqYYYYMMDDHHmmssNNN, where:
'S': 'server originated packet';
qqqq: packet sequence number;
YYYYMMDDHHmmssNNN: date and time stamp of packet transmission.

~~[0061]~~ [0066] The packet sequence numbers count from the maximum (configured by the HE command, default 19) down to zero.

~~[0062]~~ [0067] Treatment of TCP

~~[0063]~~ [0068] TCP is used in a Telnet-like session to configure the server remotely to obtain maximum flexibility. In case of a TCP connect event, the server sends the string 'password:' to the client. If the next received string via the TCP connection does not match the defined password, the server terminates the TCP connection. If the next received string matches the defined password, the server leaves the TCP connection open and sends an

overview of the commands to the user. These commands are:

HV<n>: all returned UDP packets are filled up to <n> octets (if <n> < packet size to be returned, then we do not fill the packet up, nor cut it down to prevent loss of information)

HW<n>: if within <n> milliseconds no UDP packets are received, the server starts transmitting its own packets;

HI<n> <n> defines the time in milliseconds between the server originated packets;

HE<n> <n> defines the maximum number of server originated packets.

~~[0064]~~ [0069] Each command must be given as a separate line.

~~[0065]~~ [0070] The default settings are: HV0, HW1000, HI1000, HE20.

~~[0066]~~ [0071] The set parameters are stored in memory. These could be stored in a configuration file, which loads the last parameters on reboot.

~~[0067]~~ [0072] On TCP disconnect the server terminates the session and a new login is required.

~~[0068]~~ [0073] Client part

~~[0069]~~ [0074] A user interface can offer the user three main ways to generate UDP packets:

1. With a defined time interval (uniform);
2. With a defined time interval, poisson distributed;
3. According to a packet transmission schedule file (Chirp).

~~[0070]~~ [0075] In all cases the transmitted packet size and packet interval time can be defined. In the first two cases, the packet size and interval time definition are fixed during the whole test. In the case using the packet transmission schedule file, each transmitted packet can have a different packet size and interval time, to be defined in an external file stored on disk: 'CHIRP.TXT'. An example of this file:

```
2000 300 H
700 300 H
700 300 H
700 300 H
700 300 H
700 300 H
700 300 H
700 300 H
10 30 B
10 30 B
10 30 B
10 30 B
```

Each row of this file describes one packet.

~~[0071]~~ [0076] The first column defines the waiting time before the packet is sent, the second column defines the

packet size in octets and the last column defines the first letter of the packet to be sent. Fields are delimited with a space.

~~[0072]~~[0077] Analysis of 'Chirp mode' packet sequences requires the packets to be identified. In the actual version the analysis program identifies slow streaming packets with the letter 'H' (normal packets) and buffer filling stream packets with the letter 'B' (burst packets).

~~[0073]~~[0078] The client stores every sent packet in a ring buffer slot, and marks it as 'sent'. If this ~~ringbuffer~~ring buffer slot is still filled with a frame which did not return so far, this frame is deleted from the ~~ringbuffer~~ring buffer and marked as lost in the ~~A-logfile~~A-log file.

~~[0074]~~[0079] All packets are reflected by the server, which appends a timestamp and (depending on its configuration) additional payload.

~~[0075]~~[0080] At the start of each test, two ~~logfile~~log files are opened by the client: 'AYYYYMMDDHHmmSS.TXT' and 'RYYYYMMDDHHmmSS.TXT' in which YYYYMMDDHHmmSS reflects the date and time of the start of the measurement.

~~[0076]~~[0081] The A-file contains general information on the measurement, like the content of the chirp file, the

size of the received packets from the server, and exceptions which occurred, like lost packets and reception of unidentified packets.

~~[0077]~~ [0082] The R-file contains the reception time, ~~roundtriptime~~round-trip time, one way delay time and the header letter of each matched packet.

~~[0078]~~ [0083] The client is continually alert on incoming UDP packets. Each received packet is compared with the content of the ~~ringbuffer~~ring buffer. If the packet matches, the client clears its copy in the ~~ringbuffer~~ring buffer so the ~~ringbuffer~~ring buffer slot becomes free. The client calculates the roundtrip time of the packet. If server and client are synchronized the one way delay can be calculated also, taking the server time stamp into account. The results are stored in the R-file.

~~[0079]~~ [0084] If a frame does not match with the content of the ~~ringbuffer~~ring buffer, it is stored in the A-log file as unidentified.

~~[0080]~~ [0085] Measurement procedure

~~[0081]~~ [0086] The UDP network test consists of one or more ~~testsequences~~test sequences.

~~[0082]~~ [0087] A sequence of frames is transmitted by a UDP

client over the GPRS network under test according to the content of a ~~so-called~~so-called 'chirpfile'. These frames are received by a UDP server, tagged with the reception time, extended to a defined number of octets (e.g., 700 octets), and sent back to the client.

~~[0083]~~[0088] Our chirpfile now consists of the following settings (to be read in one column)

700 300 H	700 300 H	700 300 H	700 300 H	700 300 H
700 300 H	700 300 H	700 300 H	700 300 H	700 300 H
700 300 H	700 300 H	700 300 H	10 30 B	10 30 B
10 30 B	10 30 B	10 30 B	10 30 B	10 30 B
10 30 B	10 30 B	10 30 B	10 30 B	10 30 B
10 30 B	10 30 B	10 30 B	10 30 B	6000 30 B
700 300 H	700 300 H	700 300 H	700 300 H	700 300 H
700 300 H	700 300 H	700 300 H	700 300 H	700 300 H
700 300 H	700 300 H	700 300 H	10 400 U	10 400 U
10 400 U	10 400 U	10 400 U	10 400 U	10 400 U
10 400 U	10 400 U	10 400 U	10 400 U	10 400 U
10 400 U	10 400 U	10 400 U	6000 400 U	

~~[0084]~~[0089] This file is treated as follows:

~~[0085]~~[0090] At start of the measurement 13 frames of 300 octets are sent with an interval time of 700 ms. All these frames are labeled with the letter 'H' to indicate that they are ~~primary~~primarily used to measure network latency.

The server expands these frames to 700 octets and sends them back to the client.

~~[0086]~~ [0091] This frame sequence is then followed by 17 frames of 30 octets, sent with an interval time of 10 ms. The frames within this burst are labeled with the letter 'B' to indicate their use for a downlink bandwidth measurement. The server expands these frames to 700 octets and returns them through the downlink.

~~[0087]~~ [0092] After this burst A ~~a~~ is sent, a 6000 ms pause follows. This time is used to let the network buffers empty themselves from the downlink bandwidth measurement burst.

~~[0088]~~ [0093] After this, a new frame sequence of 12 frames of 300 octets is sent, each frame with an interval time of 700 ms, to do a second network latency test. All these frames are returned by the server filled up to 700 octets.

~~[0089]~~ [0094] Finally a burst of 16 frames of 400 octets ~~are~~ is sent with an interval time of 10 ms. With this burst, uplink buffers are filled~~,~~ to measure the available uplink bandwidth. The server also expands these frames to 700 octets and sends them back. A 6000 ms pause concludes the sequence. During this time, the network may empty its buffers and prepare itself for the next test sequence.

~~[0090]~~ [0095] The sequence is continuously repeated until the operator stops it.

~~[0091]~~ [0096] If the server does not receive any further client frames ~~anymore~~ during a period of more than one second after reception of the last frame, it starts sending 20 server-originated frames with an interval of 500 ms towards the client's IP-address and port. This process stops if new frames are received from the client.

~~[0092]~~ [0097] Note: The server-originated frame process can be changed by the user. The user may change the number of frames (default 20), their interval time (default 500 ms), and the time delay after which the process starts (default 1000 ms).

~~[0093]~~ [0098] Of each frame sent by the UDP client, the label letter (packet type value) [L] as well as the transmit time (timestamp) [1] is included in the frame by the client. At reception, the server adds its reception (=transmission) time stamp [2]. When the client receives the frame back, it records its reception time [3] and calculates the ~~roundtriptime~~round-trip time [4] by subtracting [1] from [3] and the client to server time [5] by subtracting [1] from [2]. It stores from each file [3], [4] , [5] and [L] in a log file.

~~[0094]~~ [0099] Note: Client and server clocks are not (very

well) synchronized. Still, the server timestamp [2] can be used for several interesting calculations.

~~[0095]~~ [00100] In a second log file, all special events are recorded, ~~like~~ such as lost and unexpected frames. This log also contains a description of the measurement circumstances.

~~[0096]~~ [00101] Post processing calculations

~~[0097]~~ [00102] Latency measurements are used to determine quality of interactive traffic. The frames with [L]='H' are used for this purpose.

~~[0098]~~ [00103] RTT per frame : use [4] from the log file.

~~[0099]~~ [00104] Determine the client to server latency [5] of each frame, search for its minimum value, and determine for this frame a certain time value [V] for which yields: [V]:= 250 ms - [5]. [V] may be a negative value.

~~[00100]~~ [00105] Next, calculate for all frames normalized client to server time [5']:=[5]+[V].

~~[00101]~~ [00106] Note: the 250 ms value is only an example. Any user-defined value may be used here. Absolute latency results are not possible in this way, but a good impression of the dynamic performance can be obtained.

~~[00102]~~ [00107] Server to client latency [6] per frame: calculate for each frame [6]:=[4]-[5'].

~~[00103]~~ [00108] Bandwidth measurements are used to determine downlink and uplink traffic speed.

~~[00104]~~ [00109] For the downlink bandwidth calculation, the frames with [L]='B' of one burst are used.

~~[00105]~~ [00110] Suppose [DS] as the number of octets per frame, reflected by the server.

~~[00106]~~ [00111] Downlink bandwidth:= ([DS]+28 octets)*(<number of frames received from server>) / ([3]last frame-[3]first frame).

~~[00107]~~ [00112] Note: 28 octets are added to represent IP and UDP frame overhead. The calculations are made to determine bandwidth on IP-level. Therefore, frame headers should be taken into consideration.

~~[00108]~~ [00113] Note: If too few frames are received during due to frame loss, a bandwidth measurement may be discarded. For purity purposes, the number of lost frames during bandwidth measurement is reported.

~~[00109]~~ [00114] For the uplink bandwidth calculation, the frames with [L]='U' of one burst are used. Now the time

stamp that the server receives the client frames is taken into account.

~~[00110]~~ [00115] Suppose [US] as the number of octets per frame, sent by the client in the burst.

~~[00111]~~ [00116] Uplink bandwidth:=
([US]+28 octets)*(<number of frames received from server>)
/ (([3]last frame+[5]last frame)-([3]first frame+[5]first frame))

~~[00112]~~ [00117] Note: 28 octets are added to represent IP and UDP frame overhead. The calculations are made to determine bandwidth on IP-level. Therefore frame headers should be taken into consideration.

~~[00113]~~ [00118] Note: If too few frames are received during due to frame loss a bandwidth measurement may be discarded. For purity purposes the number of lost frames during bandwidth measurement is reported.

~~[00114]~~ [00119] An outage is the event that a link remains silent during a minimum predefined time.

~~[00115]~~ [00120] ~~Roundtrip~~Round-trip outages (All outages, whether uplink or downlink):
if (~~roundtriptime~~round-trip time [4]received frame - ~~roundtriptime~~round-trip time [4]former received frame) >

2000 ms,
then an outage is reported.
If between these two frames, one or more frames were lost, this is included in the outage report.

~~[00116]~~ [00121] Downlink outages:
if (latency [5']received frame - latency [5']former received frame) > 2000 ms
then an outage is reported.
If between these two frames one or more frames were lost, this is included in the outage report.

~~[00117]~~ [00122] Uplink outages:
if (latency [6]received frame - latency [6]former received frame) > 2000 ms
then an outage is reported.
If between these two frames one or more frames were lost, this is included in the outage report.

~~[00118]~~ [00123] Of each outage is reported:

time stamp
time duration
number of lost frames during the outage

~~[00119]~~ [00124] Note: The time constant can be configured by be user (default 2000 ms).

~~[00120]~~ [00125] Geographical information

~~[00121]~~ [00126] By combining the post processing calculations with simultaneously performed low level measurement by the mobile telephone (e.g., radio channel number, base station identity, mobility figures, field strength and/or radio link control) and/or geographic information (e.g., using GPS), the network quality at a geographic location becomes visible. This enables indicating poor data network quality in geographical areas, simplifying a search for weak spots in the data network.